How to Deal with Lock-Holder Preemption

Thomas Friebel

October 2008
Spinlock Basics

Spinlocks wait actively as opposed to sleeping locks

Used for short critical sections
Spinlock Wait Times – Kernbench

The diagram shows the distribution of spinlock wait times. The x-axis represents the waiting time in $2^n$ cycles, ranging from 0 to 30 cycles, while the y-axis represents the number of waits, ranging from 0 to 9000 waits.

The graph compares two scenarios:
- **Native**
- **System**

The native scenario shows a peak at 10 cycles, with a significant number of waits, while the system scenario has a broader distribution with a peak at 12 cycles.
Spinlocks and Virtualization
Spinlocks and Virtualization

How to Deal with Lock-Holder Preemption
Spinlocks and Virtualization

How to Deal with Lock-Holder Preemption

CPU 0

VCPU

30 ms
Spinlocks and Virtualization

How to Deal with Lock-Holder Preemption
Spinlocks and Virtualization

How to Deal with Lock-Holder Preemption
Is lock-holder preemption problematic?
Kernbanch in a Guest

How to Deal with Lock-Holder Preemption
Kernbench vs. 'while(true)'

How to Deal with Lock-Holder Preemption
How to Deal with Lock-Holder Preemption
### And in Numbers?

<table>
<thead>
<tr>
<th></th>
<th>guest time [s]</th>
<th>time spent spinning [s]</th>
<th>[%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>single kernbench</td>
<td>109.0</td>
<td>0.2</td>
<td>0.2%</td>
</tr>
<tr>
<td>kernbench vs while(1)</td>
<td>117.3</td>
<td>9.0</td>
<td>7.6%</td>
</tr>
<tr>
<td>difference</td>
<td>7.6%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What can we do about it?
Dealing with lock-holder preemption

LHP avoidance

• No spinlock held in userspace
• Idea: Avoid preempting guest in kernel space
• Postpone guest switch to kernel exit
• Problem: extraordinary long critical sections, e.g. Apache using sendfile()

Helping locks

• Instead of busy waiting, switch to preempted lock-holder
• Problem: finding the preempted lock-holder
Helping locks: Ingredients

1) Guest kernel: new 'yield' hypercall when waiting unusually long
   • Modify spinlock loop

2) Reasonable threshold for 'unusually long'
   • Histograms help

3) Selecting which VCPU to switch to
Threshold: Upper boundary

How to Deal with Lock-Holder Preemption
Threshold: Lower boundary

How to Deal with Lock-Holder Preemption
Scheduling Strategy

Good choices:
• VCPUs of the same VM to make progress locally
• (Potential) preempted lock-holders
• Cache-“near” VCPUs

Neither/nor:
• VCPUs in user space

Bad choices:
• VCPUs which yielded recently
What about performance?
How to Deal with Lock-Holder Preemption

Histogram with 'yield' hypercall

- LHP
- LHP yield

Number of waits vs. waiting time [2^n cycles]
## Performance

<table>
<thead>
<tr>
<th></th>
<th>wall clock [s]</th>
<th>guest time [s]</th>
<th>time spent spinning [s]</th>
<th>[%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LHP</td>
<td>34.8</td>
<td>117.3</td>
<td>9.0</td>
<td>7.6%</td>
</tr>
<tr>
<td>yield</td>
<td>33.5</td>
<td>108.4</td>
<td>0.0</td>
<td>0.0%</td>
</tr>
<tr>
<td>difference</td>
<td>-3.9%</td>
<td>-7.6%</td>
<td></td>
<td>-7.6%</td>
</tr>
</tbody>
</table>
Efficiency

557 sec. (=16x34.82)

while(true);
314 sec.

VMM + dom0
125+1 sec.

kernbench
117 sec.

How to Deal with Lock-Holder Preemption
How to Deal with Lock-Holder Preemption

Efficiency

557 sec. (=16x34.82)

VMM + dom0
125+1 sec.

8% spinning → saved
92% work → constant

while(true);
314 sec.

kernbench
117 sec.
Efficiency

- while(true); 314 sec.  
  - depends on experiment time → neutral

- VMM + dom0 125+1 sec.  
  - mainly induced by kernbench work part → constant

- kernbench 117 sec.  
  - 8% spinning → saved
  - 92% work → constant

557 sec. (=16×34.82)
Efficiency

VMM + dom0
125+1 sec.

mainly induced by
kernbench work part
→ constant

8% spinning
→ saved

\[
\frac{117 \text{ sec}}{117 \text{ sec} + 126 \text{ sec}} \times 7.6\% = 3.7\%
\]

➔ Real result of 3.9% is reasonable

➔ Highly efficient
FIFO ticket spinlocks
FIFO ticket spinlocks

Next ticket in dispenser: queue tail

„Now serving“ display at counter: queue head

Lock: atomic( ticket = tail++ ); while ( head != ticket );

Unlock: atomic( head++ );
**FIFO ticket spinlocks**

- **CPU 0**: VCPU
- **CPU 1**: ticket: 1
- **CPU 2**: t:2
- **CPU 3**: ticket: 3

30 ms
## Ticket locks and virtualization

<table>
<thead>
<tr>
<th></th>
<th>wall clock [s]</th>
<th>guest time [s]</th>
<th>time spent spinning [s]</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>LHP</td>
<td>2825.1</td>
<td>22434.2</td>
<td>22270.4</td>
<td>99.3%</td>
</tr>
</tbody>
</table>
# Ticket locks and virtualization

<table>
<thead>
<tr>
<th></th>
<th>wall clock [s]</th>
<th>guest time [s]</th>
<th>time spent spinning [s]</th>
<th>yield [s]</th>
<th>time spent spinning [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LHP</td>
<td>2825.1</td>
<td>22434.2</td>
<td>22270.4</td>
<td>34.1</td>
<td>99.3%</td>
</tr>
<tr>
<td>yield</td>
<td>34.1</td>
<td>123.6</td>
<td>6.6</td>
<td>5.4%</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

Lock-holder preemption quite serious: 7.6% guest time wasted

Helping locks:
3.9% system performance improvement! (Amdahl's law explains why)

New ticket spinlocks:
30 secs kernbench takes 45 minutes

Helping locks help here, too
Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2008 Advanced Micro Devices, Inc. All rights reserved.